Finding Orthogonal Pairs using Rosetta (Linux)

Table of Contents

1 Preparing a the PDB for use in Rosetta

For extra information:
https://www.rosettacommons.org/docs/latest/rosetta_basics/preparation/preparing-structures

1.1 Materials

- Protein DataBase (PDB) file(s) of your desired complex
- Openbabel Software package

1.2 Cleaning

- Cleaning PDB files before use in Rosetta is essential, because PDB files contain information that Rosetta doesn't need and might even crash it. Most PDBs can be cleaned with *clean_pdb.py*, but when a molecule is present that is unknown to Rosetta, but relevant for your protein complex (e.g. a small molecule or synthetic amino acid), more steps are needed (see 1.3)
- Use *~rosetta/main/source/src/apps/public/relax_w_allatom_cst/clean_pdb_keep_ligand.py <your_pdb> - ignorechain*
- **NOTE:** after every command > <name_logfile> can be pasted to save whatever the command prints to the terminal.

1.3 Introducing unrecognized residues

- Often the use of non-standard residues, like small molecules, is necessary for the formation of the protein complex. These residues need to be added to the Rosetta database/your simulation.
- Extracting the Residues from your PDB file
  - For each Residue, replace the PDB chain name with one that is not present in the PDB
  - Extract the residue into a separate PDB file:
  - grep '<residue> <chain>' <pdb> > <residue>.pdb
- Making .mol2 files for each of the extracted PDB files
  - babel <residue>.pdb <residue>.mol2
- Generate .params and .pdb files from the .mol2 files:
  - *~rosetta/main/source/scripts/python/public/molfile_to_params.py –n <res> -p <res> <residue>.mol2*
  - NOTE: be sure to name the resulting files (-n<> and –p<>) with the 3-letter code used in the complete PDB
  - For multiple .params at once you can use *batch_molfile_to_params.py*
- Replace the lines for the unrecognized residues in the original (cleaned) PDB with the lines from the generated PDBs (specifically for each residue)

1.4 Relaxing

- In order for your PDB to be usable by Rosetta it needs to be relaxed into the Rosetta forcefield, not relaxing your PDB can lead to steric clashes and may result into faulty results and crashing simulations.
- Make a flags file containing the commands for the simulation (at least 10 structures recommended) (see appendix 5.1 for an example)
- Run the simulation
  - *~rosetta/main/source/bin/relax.default.linuxgccrelease @<flags>*

- The result is a PDB ready for use in rosetta.
- The lower the score of the structure, the more stable it is.

2 Backrub Simulation

Backrub simulations are often used to further stabilize flexible proteins by introducing backbone flexibility, it rotates the backbones of selected residues respective to its neighbours to find a state of reduced Gibbs free energy. For extra information on the backrub simulation and creating additional files: https://www.rosettacommons.org/docs/latest/application_documentation/structure_prediction/backrub

2.1 Materials

- Relaxed PDB(s)
- List of pivot residues (default = all)
  - This list specifies which residues of the PDB you want to perform a backrub on
- Flags file, for an example see appendix 5.2
- (optional) a resfile, specifying which residues should have their side chains repacked
  - A resfile should be structured like this:
    <residue number> <chain> <command>:
    NATRO
    Start
      3 A NATAA
      7 A NATAA
    Etc.
  - For more info on creating resfiles: https://www.rosettacommons.org/docs/latest/rosetta_basics/file_types/resfiles
- (optional) the .params files used in the relaxation simulation

2.2 Protocol

- Use *~rosetta/main/source/bin/backrub.default.linuxgccrelease @flags*
- The outputs are the last PDB generated and the PDB with the lowest score for <number of constructs>
- The lower the score, the more stable the structure

There is also a public server available for backrub simulations, https://kortemmelab.ucsf.edu/backrub/, but this limits your control over your simulation, e.g. not being able to specify which residues to backrub

3 Computational Alanine Scanning

In order to find which residues are important for the binding in your protein complex, Alanine scanning is performed, Alanine is an amino acid that often does not contribute to the binding between proteins, thus an alanine scan finds the difference in Gibbs free energy, indicating the importance of the native amino acids. This protocol makes use of the Robetta Server for Computational Alanine Scanning. Robetta Server: http://robetta.bakerlab.org/

3.1 Materials

- Relaxed/Backrubbed PDB(s)
- (optional) mutations list
  - A mutations list should be structured like this:
    <residue number> <chain id>:
    3 A
    7 A
    12 A
    123 B
    Etc.

3.2 Robetta Server

- The Robetta server provides Computational Alanine scans, it uses a PDB and a defined interface, which chains of your PDB react with which, and optionally a list of residues to mutate. Without the list, the server will determine which residues are important by itself.
- What the Robetta server does:
  - Add polar hydrogens.
  - Optimize the hydrogen bonds using a Monte Carlo algorithm.
  - Define interface residues: uses the list of mutations that is submitted, otherwise an interface residue is defined as a residue with an atom within a sphere with a radius of 4 Å of an atom of the binding partner, or a residue that becomes significantly buried when the complex forms.
  - Mutates all interface residues one by one into an alanine and repacks (side chains are moved, but backbone stays fixed) every sidechain with at least 1 atom in a sphere of 5 Å around the mutated residue.
  - Computes the difference of free Gibbs energy between the wildtype and mutated complex.

3.3 Processing results

- The output of the server simulation are a bunch of files, of which the .results file is the most important, as these contain the scores.
- Scores can be easily visualized by plotting them in a heatmap, in a 2D map or in pyMol on the surface structure of the Protein.
- The residues that are important for the binding interface should be used to look for orthogonal pairs

4 Point Mutant scanning

To find an orthogonal binding between A and B, the interface must first be destabilized and then be restabilized. Then additional orthogonality needs to be checked. For additional information on the point mutant scan application and creating additional files: https://www.rosettacommons.org/docs/latest/application_documentation/design/pmut-scan-parallel

4.1 Destabilizing the binding interface

- In order to introduce the highest possible orthogonality, the binding must first be destabilized by introducing a mutation in partner A, for which the Gibbs free energy increases significantly. If the increase is not high enough, the mutated A can still bind to partner B
- Materials:
  - Backrubbed PDB(s)
  - Essential flags:
    - *-output_mutant_structures*
      This command will output the PDB for any mutation that (de)stabilizes the interface by better than a ddG cutoff (default = -1, can be changed using –ddG_cutoff), these PDBs will be used in the next step.
    - ***-alter_spec_disruption_mode***
      This command will tell the program to look for **destabilizing** mutations instead of stabilizing mutations, however this does **not** reflect on the resulting log file, so – should be + and vice versa.
  - flags file (see appendix 5.3 for an example)
  - (optional, but strongly recommended) mutations list
    - A mutations list should be structured like this:
      <chain> <old amino acid> <residue number> <new amino acid>:
      A E 19 R #single point mutant
      A E 19 R Q K 943 D #double point mutant

- Protocol
  - ~rosetta/main/source/bin/pmut_scan_parallell.linuxgccrelease @pmsaflags **> pmsaoutput.txt**
- Output:
  - Log file: pmsaoutput.txt
    - Logfile consists of:
      <mutation> <mutation PDB numbering> <ddG change> <total structure energy>
  - PDBs with every destabilizing mutation

4.2 Restabilizing the binding interface

- In order to restabilize binding to the mutated partner A, partner B also has to be mutated. The ddG decrease should be high enough to ensure that binding between the mutated pairs occurs, if this is not the case the restabilizing effect is not good enough and the mutated pair won't bind.
- Materials:
  - PDBs with a mutation
  - Flags file (same as in step 4.2, but:)

- Without alter_spec_disruption_mode this time
- Output_mutant_structures optional (this simulation has to be run a lot of times, so generating too many PDBs will clog your computer, desired PDSs can be generated a posteriori by running the simulation with this option)

    - (optional, but strongly recommended) mutations list (similar to step 4.1)
- Protocol
    - ~rosetta/main/source/bin/pmut_scan_parallell.linuxgccrelease @pmsaflags **> pmsaoutput.txt**
    - (optional) combine the best single point mutations into a mutant list with double point mutations and run the simulation again with this list
- Output:
    - Log file: pmsaoutput.txt (similar to step 4.1)
    - (optional) PDBs with the new interfaces

4.3 Checking the orthognality

- If the above steps have succeeded, you have now checked whether the mutated A does not bind to the normal B, and that the binding between mutated A and mutated B is strong enough. The binding you still need to check is the orthogonality between the normal A and mutated B.
- Materials
    - PDB(s) **without** mutation
    - Flags file (see appendix 5.3 for an example)
    - **Mutations list** (similar to step 4.1)
- Protocol
    - Make your mutations list based on the results of step **4.2**
    - ~rosetta/main/source/bin/pmut_scan_parallell.linuxgccrelease @pmsaflags **> pmsaoutput.txt**
- Output:
    - Log file: pmsaoutput.txt (similar to step 4.1)
    - (optional) PDBs with the compensating mutations

4.4 Additional checks

If your protein complex contains more than 2 proteins, make sure that creating an orthogonal pair does not disturb the other proteins in your complex. For example if A and B form a dimer and C binds to B, make sure that when you create an orthogonal pair between B and C no residues in the binding interface of A and B are mutated.

5 Appendix

5.1 Example of a relax flags file

```
-database ../Rosetta/main/database
-extra_res_fa <.params1> <.params2> etc.
-relax:constrain_relax_to_start_coords
-relax:coord_constrain_sidechains
-relax:ramp_constraints false
-s <your_pdb> or –l <your_list_of_pdbs>
-ex1
-ex2
-use_input_sc
-flip_HNQ
-no_optH false
-nstruct <number oc constructs>          #takes roughly an hour per structure for
                                         every 680 amino acids
```

5.2 Example of a backrub flags file

```
-database ~rosetta/main/database/
-in
 -file
  -s <your pdb> or –l <your list of pdbs>
 -ignore_unrecognized_res
- extra_res_fa <.params1> <.params2> etc.
-backrub

 -ntrials <number of tries per construct> #recommended: 10000

-nstruct <number of constructs> #recommended: 5-50
-resfile <your resfile>
-pivot_residues <your list of pivot residues> #e.g. 32 45 123 473 etc.
```

5.3 Example of a point mutant scan flags file

```
-s <pdb1> <pdb2> etc. or –l <your list of pdbs>
-database ~rosetta/main/database/

-double_mutant_scan          #with this command the protocol checks both 1 and 2
                             compensating mutants
(optional) –mutant_list <file>      # only makes specified mutants
-output_mutant_structures           # creates PDB for every stabilizing mutation
-alter_spec_disruption_mode         # makes the protocol look for destabilizing
mutations
-ex1
-ex2
-use_input_sc
-flip_HNQ
-no_optH false
-ignore_unrecognized_res
-no_his_his_pairE
-ddG_cutoff <value>
```